Tutorial

Table of contents

1. Overview

DIDLTools is a Java toolkit for the construction, validation, serialization and de-serialization for MPEG-21 DIDL data model. This tutorial exists to help newcomers quickly understand the abilities of the DID-API and existing implmentations. The tutorial should provide basic working knowledge of the DIDLTools and the DIDL structure it supports. A sample application will be created to illustrate the various elements necessary to create a DID.

2. Modeling a DIDL Structure

Before we start coding we need to define the structure of our digital objects according to the DIDL specification. Each DID is comprised of a number of key elements, these elements include:

- **Resource** - A resource is an individually identifiable asset such as a video or audio clip, an image, or a textual asset. A resource may also potentially be a physical object. All resources must be locatable via an unambiguous address using By-Reference or stored By-Value.
- **Component** - A component is the binding of a resource to all of its relevant descriptors. These descriptors are bits of information related to all or part of the specific resource instance. Component descriptors will typically contain control or structural information about the resource (such as bit rate, character set, start points or encryption information) but generally not information describing the "content" within.
- **Descriptor** - A descriptor associates information with the enclosing element. This information may be a component (such as a thumbnail of an image) or a portion of text.
- **Statement** - A statement is a literal textual value that contains information but not an asset. Examples of likely statements include descriptive, control, revision tracking, or identifying information.
- **Item** - An item is a grouping of sub-items or components that are bound to relevant descriptors. Items may contain choices, which allow them to be customized or configured. Items themselves may be conditional. If an item contains no sub-items, then it can be called an entity. If it contains sub-items, then it can be called a compilation.

For our first example, lets say we have a MARC-XML document which we'd like to package as a DID. Our MARC-XML document has a doi (i.e. info:doi/xx.xxxx/j.dyepig.2004.06.022") which we'll use for our content identifier. For our digital object, we'll generate a UUID and prefix this identifier with a local info URI (i.e. info:lanl-repo/i/dd7b17ea-bddf-11d9-9de5-c11b6cd8559). Our first DID will be composed for the following:

- **Resource** -> MARC-XML Document stored as inline XML, know as referenced

"By-Value".
- **Component** -> Encapsulates Resource and its Descriptor, which defines elements such as copyright and usage.
- **Descriptor** -> Enclosed Statement contains Content Type defining adminstrative metadata for digital object. In this use case, we will create a new content type, see "Modeling DIDL Content Type" below for additional information.
- **Item** -> Encapsulates all resources and metadata pertaining to this digital object.

Figure 1

The DIDL Specification and DID-API support complexity far beyond the model defined above. Starting with a simple model allows us to explore the key elements without any unnecessary complexity. For those who wish to jump ahead, look into the MyComplexDidl implementation provided with DIDExamples distribution.

3. Modeling a DIDL Content Type

Each Descriptor contains a child element named Statement. The Statement is an open-ended XMLSchema element allowing data from any namespace. In order to properly support the DIDL schema's flexibility, the DID-API provides an interface to support the construction, serialization, and de-serialization of virtually any metadata structure. These metadata structures are referred to as Content Types within the the DID-API. For each content type we need to define a base class, a serializer implementation, and a de-serializer implementation. For this tutorial, we'll create a simple content type containing the following attributes:

- **id** -> A unique identifier of the parent for which the content type represents.
- **resourceURI** -> A resolvable resource uri for a resource.
- **copyright** -> A brief note indicating copyright ownership.
- **usage** -> A brief note indicating usage restrictions.

```
The XML serialized MyContent object will be of the form:
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="resource" type="xs:string"/>
   <xs:element name="copyright" type="xs:string"/>
   <xs:element name="usage" type="xs:string"/>
   <xs:attribute name="id" type="xs:ID"/>
 </xs:schema>
```

4. Creating your first DIDL

Note: Source code for this tutorial is available within the DIDExamples distribution.

To implement the DIDL model we've structured above, we need to create the following classes:

- MyContent - MyContent Content Type Object
- MyContentSerializer - Generates XML String from MyContent object
- MyContentDeserializer - Produces MyContent object from MyContent XML fragment
- MySimpleComponent - Defines the structure of our Component
- MySimpleDidl - Defines the structure of our DIDL
- MySimpleDidlTest - Test Harness

5. MyContent

First we need to model our content type object. To do this we'll create a class named MyContent. Each content type requires three components; a data object, a serializer, and a deserializer. This class is an example of the data object. The serializer and deserializers are located in the org.foo.didl.serialize package, named MyContentSerializer and MyContentDeserializer, respectively. The three components may also be implemented as a single class.

Implementing the model we defined above will produce a class similar to the following:

```
public class MyContent {
    public static final String NAMESPACE =
"http://mysimplecontent.com/ns/MySimpleContent/";
    public static final String PREFIX = "cn";
    private String id;
    private String copyright;
    private String usage;
    private String resourceUri;

    public String getCopyright() {
        return copyright;
    }
    public void setCopyright(String copyright) {
        this.copyright = copyright;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getResourceUri() {
        return resourceUri;
    }
    public void setResourceUri(String resourceUri) {
        this.resourceUri = resourceUri;
    }
    public String getUsage() {
        return usage;
    }
    public void setUsage(String usage) {
        this.usage = usage;
    }
}
```

In addition to the object fields we also added a couple constants to define the XML namespace.

6. MyContentSerializer

Next, we need to define how an instance of our content type should be serialized. To do this, we'll create a class named MyContentSerializer. This class will need to implement the DIDLSerializerType interface which is called during the DIDL creation process. During DIDL Serialization a content type registry is checked for a implementation for the current object. Provided there's a match, the DIDL serialization interface will delegate to the implementing class. MyContentSerializer is an example of an implementing class. In MySimpleDidl we'll register this implementation in getXML(). In the more complex tutorial, MyComplexDidl, we call MySerializationFactory and ask it to return a DIDLSerializerType containing the registered content type implementations.

There are a number of ways in which we can serialize our object to XML, but for this example we'll just pull out the information we need and write the xml form to a PrintWriter.

```

public class MyContentSerializer implements DIDLSerializerType {

    // Implements DIDLSerializerType.write()
    public void write(OutputStream stream, Object obj)
            throws DIDLSerializationException {
        MyContent msc = (MyContent) obj;
        try {
            PrintWriter out = new PrintWriter(stream, true);
            out.print("<" + MyContent.PREFIX + ":content xmlns:" +
MyContent.PREFIX + "=\"" + MyContent.NAMESPACE + "\"");
            out.print(" id=\"" + msc.getId() + "\">");

            if (msc.getResourceUri() != null)
                out.print("<" + MyContent.PREFIX + ":resource>" +
msc.getResourceUri()
                        + "</" + MyContent.PREFIX + ":resource>");
            if (msc.getCopyright() != null)
                out.print("<" + MyContent.PREFIX + ":copyright>" +
msc.getCopyright()
                        + "</" + MyContent.PREFIX + ":copyright>");
            if (msc.getUsage() != null)
```

```
                out.print("<" + MyContent.PREFIX + ":usage>" +
msc.getUsage() + "</" + MyContent.PREFIX + ":usage>");

            out.print("</" + MyContent.PREFIX + ":content>");
            out.close();
        } catch (Exception ex) {
            throw new DIDLSerializationException(ex.getMessage());
        }
    }
}
```

The write() method will be invoked when a higher level request for serialization is made. The registry will tell the application it should use this implmentation for the MyContent content type and the resulting data will be written to the output stream.

7. MyContentDeserializer

Now that we've defined our content type's serialization form we can implement our de-serializer. Next, we'll create a new class, named MyContentDeserializer, to demonstrate how XML serialized content can be deserialized. The class implements the DIDLDeserializerType interface which is called during the DIDL parsing process.

For this example, we'll do the following:

1. Read InputStream to extract XML Content as a String
2. Initialize our XPathProcessor
3. Set MyContent fields using XPath queries
4. Return MyContent object

```

public class MyContentDeserializer implements DIDLDeserializerType {
    ArrayList<String> nodes;

    // Implements DIDLDeserializerType.read()
    public Object read(InputStream in) throws DIDLSerializationException {
        MyContent msc = new MyContent();
        try {
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            byte[] buffer = new byte[1024];

            int len = 0;
            while ((len = in.read(buffer)) != -1) {
                bout.write(buffer, 0, len);
            }
            bout.close();

            String xml = bout.toString();

            XPathProcessor xpp = new XPathProcessor();
            xpp.setDocument(xml);
            xpp.addNamespace(MyContent.NAMESPACE, MyContent.PREFIX);
            msc.setId(xpp.xpath("//@id").get(0));
            // Check for resource elements
            nodes = xpp.xpath("//" + MyContent.PREFIX + ":resource");
            if (nodes.size() > 0)
                msc.setResourceUri(nodes.get(0));
            // Check for copyright elements
            nodes = xpp.xpath("//" + MyContent.PREFIX + ":copyright");
            if (nodes.size() > 0)
                msc.setCopyright(nodes.get(0));
            // Check for usage elements
            nodes = xpp.xpath("//" + MyContent.PREFIX + ":usage");
            if (nodes.size() > 0)
                msc.setUsage(nodes.get(0));

            return msc;
        } catch (IOException e) {
            throw new DIDLSerializationException(e.getMessage());
        } catch (Exception e) {
            throw new DIDLSerializationException(e.getMessage());
        }
    }
}
```

Thus far, we've defined our content type, MyContent, how it can be written as XML and how to parse the XML to create a MyContent object. Next we need to define the structure of our component.

8. MySimpleComponent

DIDL Componets wrap up all the information about a resource into a clean package. A Component can contain any number of Descriptor elements. Descriptor elements are essentially wrappers for our content type information. A DIDL Component may all contain any number of Resource elements, provided they all relate to the same resource. In MySimpleComponent we'll define a simple didl component implementation. The create() and parse() will be used to construct and deconstruct the component.

```

      This example will produce a Component similar to the structure
outlined below:
      <didl:Component>
        <didl:Descriptor>
           <didl:Statement>
              <content>...</content>
           </Statement>
        </didl:Desciptor>
        <didl:Resource mimeType="..." ref="..."/>
        <didl:Resource mimeType="...">...</didl:Resource>
      </didl:Component>
```

For this example, we'll do the following:

1. Define fields and associated getter/setters for transitory information.
2. Define create() method implementing previously defined Component structure.
3. Define parse() method to deconstruct our Component structure.

```

public class MySimpleComponent {
    private String id;
    private String mimetype;
    private URI resourceURI;
    private String content;
    private String copyright;
    private String usage;

    public MySimpleComponent() {
    }

    public MySimpleComponent(String id, String mimetype, String copyright,
            String usage, URI resourceUri, String content) {
        this.setId(id);
        this.setMimetype(mimetype);
        this.setResourceURI(resourceUri);
        this.setCopyright(copyright);
        this.setUsage(usage);
        this.setContent(content);
    }

    public ComponentType create(DIDLType didl) throws MyDidlException {
        ComponentType com = didl.newComponent();
        com.setId(Identifier.createXMLIdentifier());

        // Set MyContent
        MyContent mc = new MyContent();
        mc.setId(id);
        if (resourceURI != null) {
            try {
                mc.setResourceUri(resourceURI.toURL().toString());
            } catch (MalformedURLException e) {
                throw new MyDidlException(e.getMessage(), e);
            }
        }
        mc.setCopyright(copyright);
        mc.setUsage(usage);

        StatementType stmt = didl.newStatement();
        stmt.setMimeType("application/xml; charset=utf-8");
        stmt.setContent(mc);
        com.addDescriptor(didl.newDescriptor()).addStatement(stmt);

        if (getResourceURI() != null) {
            ResourceType resource = didl.newResource();
            resource.setMimeType(getMimetype());
            resource.setRef(getResourceURI());
            try {
                resource.setContent(new
ByteArray(getResourceURI().toURL()));
            } catch (MalformedURLException e) {
                throw new MyDidlException(e.getMessage(), e);
            }
            com.addResource(resource);
        }

        if (getContent() != null) {
            ResourceType resource = didl.newResource();
            resource.setMimeType(getMimetype());
            resource.setContent(new ByteArray(getContent()));
            com.addResource(resource);
        }

        return com;
    }

    public static MySimpleComponent parse(ComponentType com) throws
Exception {
        MySimpleComponent msc = new MySimpleComponent();

        for (DescriptorType desc : com.getDescriptors()) {
            Object content = desc.getStatements().get(0).getContent();
            if (MyContent.class.isInstance(content)) {
                MyContent mc = (MyContent) content;
                msc.setId(mc.getId());
                msc.setCopyright(mc.getCopyright());
                msc.setUsage(mc.getUsage());
            }
        }

        for (ResourceType r : com.getResources()) {
            msc.setResourceURI(r.getRef());
            msc.setMimetype(r.getMimeType());

            if (msc.getResourceURI() == null) {
                ByteArray ba = (ByteArray) (r.getContent());
                msc.setContent(ba.getString());
            }
        }

        return msc;
    }

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getCopyright() {
        return copyright;
    }
    public void setCopyright(String copyright) {
        this.copyright = copyright;
    }
    public String getUsage() {
        return usage;
    }
    public void setUsage(String usage) {
        this.usage = usage;
    }
    public URI getResourceURI() {
        return resourceURI;
    }
    public void setResourceURI(URI resourceURI) {
        this.resourceURI = resourceURI;
    }
    public String getMimetype() {
        return mimetype;
    }
    public void setMimetype(String mimetype) {
        this.mimetype = mimetype;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
```

Now that we've defined our Component, we can take the next step and define our DIDL structure.

9. MySimpleDidl

Next we'll wrap up our content types and components into DIDL Items and implement the DIDL structure we previously defined.

```

  This example will produce a DIDL similar to the structure outlined below:
  <didl:DIDL>
    <didl:Item>
<didl:Descriptor><didl:Statement>...</didl:Statement></didl:Desciptor>
      <didl:Component>
<didl:Descriptor><didl:Statement>...</didl:Statement></didl:Desciptor>
        <didl:Resource mimeType="..." ref="..."/>
        <didl:Resource mimeType="...">...</didl:Resource>
      </didl:Component>
    </didl:Item>
  </didl:DIDL>
```

For this example, we'll do the following:

1. Define fields and associated getter/setters for transitory information.
2. Define create() method implementing previously defined DIDL structure.
3. Define parse() method to deconstruct our DIDL structure.
4. Define method to get XML serialization of our DIDL object

```
public class MySimpleDidl {
    public static final String DEFAULT_COPYRIGHT = "Copyright (c)
2004-2006, Los Alamos National Laboratory.";
    public static final String DEFAULT_USAGE = "Public granted rights to
use, reproduce, and distribute this information";
    private String did;
    private String contentId;
    private String copyright;
    private String usage;
    private MySimpleComponent myComponent;

    public MySimpleDidl() {
    }

    public void setComponent(String id, String mimetype, URI resourceUri,
            String content) {
        myComponent = new MySimpleComponent(id, mimetype, copyright, usage,
                resourceUri, content);
    }

    public String getXML() throws Exception {
        DIDLType didl = create();
        ByteArrayOutputStream stream = new ByteArrayOutputStream();
        DIDLSerializer serializer = new DIDLSerializer();
        serializer.getRegistry().addSerializer(MyContent.class,
                MyContentSerializer.class);
        serializer.write(stream, didl);
        return stream.toString();
    }

    private DIDLType create() throws MyDidlException {
        try {

            // Create a new DIDL
            DIDLType didl = (DIDLType) new DIDL();

            // Set the DIDL Document Attribute as a URI
            didl.setDIDLDocumentId(new URI(did));

            // Create a new Item and set the unique identifier
            ItemType item = didl.newItem();
            item.setId(Identifier.createXMLIdentifier());
            // Create a new MyContent object for the first item
            MyContent mc = new MyContent();
            mc.setId(contentId);
            mc.setCopyright(copyright);
            mc.setUsage(usage);

            // Create a Statement and set MyContent
            StatementType stmt = didl.newStatement();
            stmt.setMimeType("application/xml; charset=utf-8");
            stmt.setContent(mc);
            item.addDescriptor(didl.newDescriptor()).addStatement(stmt);

            // Create a metadata item
            ItemType rootItem = didl.addItem(item);

            // Add the metadata-by-value components
            rootItem.addComponent(myComponent.create(didl));

            return didl;
        } catch (Exception ex) {
            throw new MyDidlException("error in create MySimpleDidl
record", ex);
        }
    }

    public void parse(java.io.InputStream stream) throws MyDidlException {
        try {
            // Initialize a DIDLDeserializer to which we can register new
            // content handlers
            DIDLDeserializer deserializer = new DIDLDeserializer();

            // Register MyContent Deserializer & Content Strategy
            deserializer.getRegistry().addDeserializer(MyContent.class,
                    MyContentDeserializer.class);
            deserializer.getStrategy().addContentStrategy(
                    new SimpleContentCondition(null, null,
MyContent.NAMESPACE,
                            MyContent.class));

            // Pass the InputStream to the deserializer to obtain the
parsed
            // DIDL
            DIDLType didl = (DIDLType) deserializer.read(stream);

            // Obtain DIDLDocumentID from the parsed DIDL
            did = didl.getDIDLDocumentId().toString();

            // Obtain MyContent object from parsed didl and set associated
            // values
            MyContent mc = (MyContent)
(didl.getItems().get(0).getDescriptors()
                    .get(0).getStatements().get(0).getContent());
            contentId = mc.getId();
            if (mc.getCopyright() != null)
                copyright = mc.getCopyright();
            if (mc.getUsage() != null)
                usage = mc.getUsage();

            // Obtain Component and parse object to create a
MySimpleComponent
            // object
            myComponent = MySimpleComponent.parse(didl.getItems().get(0)
                    .getComponents().get(0));

        } catch (Exception ex) {
            ex.printStackTrace();
            throw new MyDidlException("error in parse MySimpleDidl", ex);
        }
    }

    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append("did: ")
        .append(did)
        .append("\ncontentid: ")
        .append(contentId);
        return sb.toString();
    }

    public String getContentId() {
        return contentId;
    }
    public void setContentId(String contentId) {
        this.contentId = contentId;
    }
    public String getDocumentId() {
        return did;
    }
    public void setDocumentId(String did) {
        this.did = did;
    }
    public MySimpleComponent getMyComponent() {
        return myComponent;
    }
    public String getCopyright() {
        return copyright;
    }
    public void setCopyright(String copyright) {
        this.copyright = copyright;
    }
    public String getUsage() {
        return usage;
    }
    public void setUsage(String usage) {
        this.usage = usage;
    }
}
```

10. MySimpleDidlTest

MySimpleDidlTest provides a test harness for MySimpleDidl.

```
public class MySimpleDidlTest {

    private static String outfile;

    public static void main(String[] args) throws Exception {
        if (args.length > 0)
           outfile = args[0];

        MySimpleDidl com = new MySimpleDidl();
        com.setContentId("info:doi/xx.xxxx/j.dyepig.2004.06.022");
com.setDocumentId("info:lanl-repo/i/dd7b17ea-bddf-11d9-9de5-c11b6cd8559");
        com.setCopyright("Copyright (c) 2004-2006, Your Institution");
        com.setUsage("Example has been released to the public domain");
com.setComponent("info:lanl-repo/ds/99569f9a-23db-4eb2-a0d4-ec1ff34dae5f",
                         "application/xml",
                          new
URI("http://www.loc.gov/standards/marcxml/Sandburg/sandburg.xml"),
MyDidlUtils.fetch("http://www.loc.gov/standards/marcxml/Sandburg/sandburg.xml"));

        // Output to file?
        if (outfile != null) {
           FileOutputStream fos = new FileOutputStream(new File(outfile));
           fos.write(com.getXML().getBytes());
        }

        // Create a second instance from the first
        MySimpleDidl com2 = new MySimpleDidl();
        com2.parse(new ByteArrayInputStream(com.getXML().getBytes()));
        System.out.println(com2.getXML());
        }
}
```